WYK 04

DJANGO

WPROWADZENIE DO DJANGO

Darmowy, przede wszystkim oparty na OpenSource framework na python

Co oznacza framework:

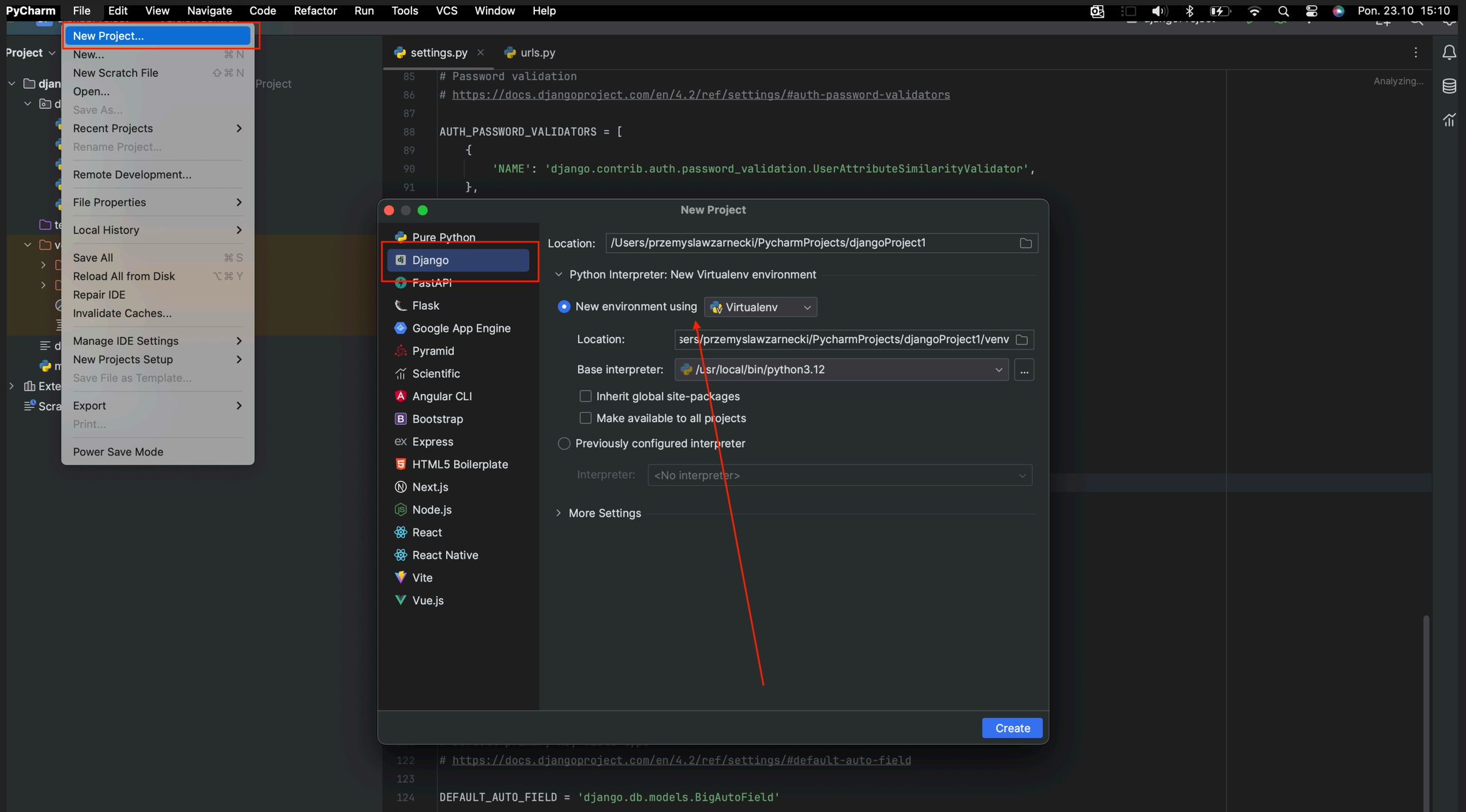
1. **Szybkość:** Django oferuje wiele gotowych rozwiązań, dzięki czemu tworzenie aplikacji jest szybsze niż w przypadku pisania kodu od podstaw.
2. **Bezpieczeństwo:** Django zapewnia wiele wbudowanych mechanizmów bezpieczeństwa, takich jak ochrona przed atakami CSRF, SQL injection i XSS.
3. **Skalowalność:** Django jest skalowalny i może obsługiwać duże aplikacje.
4. **Społeczność:** Django ma dużą i aktywną społeczność, która ciągle rozwija framework i tworzy nowe narzędzia.

WPROWADZENIE DO DJANGO

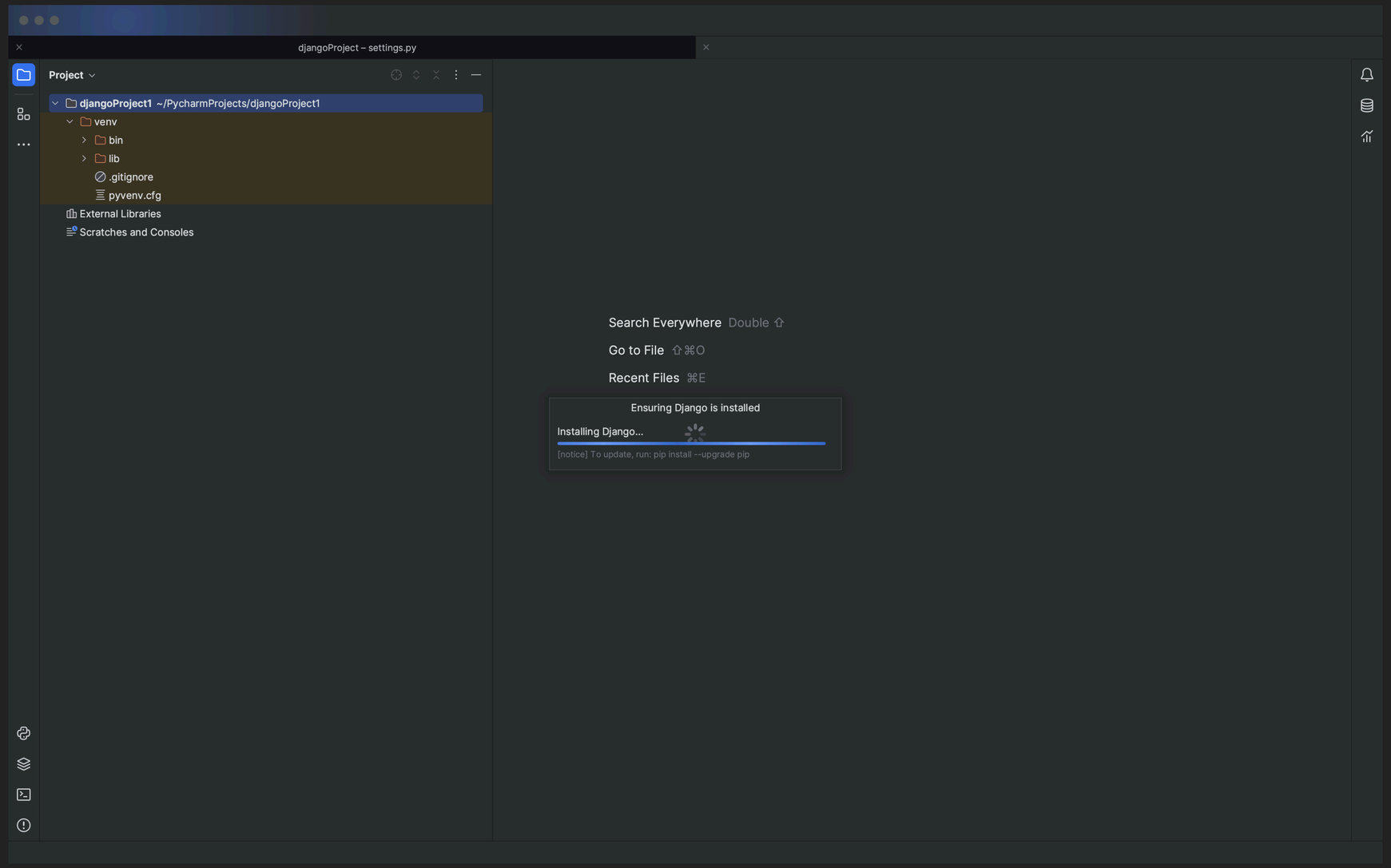
Są oczywiście minusy

1. **Full stack -** rozwiązania są dość szablonowe - problem dla tych, co chcą kombinować
2. **Wydajność:** zdaniem niektórych nieco wolniejszy od Flask
3. **Rozmiar:** Gigantyczna ilość elementów. Dużo do nauki
4. **Python -** na początku może być nieco problemów z odpaleniem projektu. Przynajmniej nie na swoich komputerach

JAK ZACZĄĆ


PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help
Pon. 23.10 15:10
New Project...
New... ⌘N
New Scratch File ⇧⌘N
Open...
Save As...
Recent Projects
Rename Project...
Remote Development...
File Properties
Local History
Save All ⌘S
Reload All from Disk ⌥⌘Y
Repair IDE
Invalidate Caches...
Manage IDE Settings
New Projects Setup
Save File as Template...
Export
Print...
Power Save Mode
settings.py
urls.py
Analyzing...
# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
{
'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
},
New Project
Pure Python
Django
FastAPI
Flask
Google App Engine
Pyramid
Scientific
Angular CLI
Bootstrap
Express
HTML5 Boilerplate
Next.js
Node.js
React
React Native
Vite
Vue.js
Location: /Users/przemyslawzarnecki/PycharmProjects/djangoProject1
Python Interpreter: New Virtualenv environment
New environment using Virtualenv
Location: sers/przemyslawzarnecki/PycharmProjects/djangoProject1/venv
Base interpreter: /usr/local/bin/python3.12
Inherit global site-packages
Make available to all projects
Previously configured interpreter
Interpreter: <No interpreter>
More Settings
Create
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

POCZĄTEK


djangoProject – settings.py
Project
djangoProject1 ~/PycharmProjects/djangoProject1
venv
bin
lib
.gitignore
pyvenv.cfg
External Libraries
Scratches and Consoles
Search Everywhere Double ⇧
Go to File ⇧⌘O
Recent Files ⌘E
Ensuring Django is installed
Installing Django...
[notice] To update, run: pip install --upgrade pip

djangoProject1 Version control djangoProject1

djangoProject – settings.py djangoProject1 – urls.py

Project

djangoProject1 ~/PycharmProjects/djangoProject1
djangoProject1
templates
venv
bin
lib
.gitignore
pyvenv.cfg
manage.py
External Libraries
Scratches and Consoles

settings.py urls.py

```
"""
URL configuration for djangoProject1 project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
import ...

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

do różnych zmian używamy terminala - nie konsoli pythona

POWRÓT DO TERMINALA

1. Tworzenie środowiska wirtualnego Pythona
2. W folderze wpisujemy python3 -venv venv
3. Podlinkowujemy środowisko wirtualne
    1. Linux: source venv/bin/activate
    2. Windows: venv\Scripts\activate
    3. Teoretycznie powinno nam zadziałać później w terminalu code .

POWRÓT DO TERMINALA

4. Instalacja Django

W zależności od wersji pythona pip albo pip3:

pip3 install django

Można użyć pip install -m django

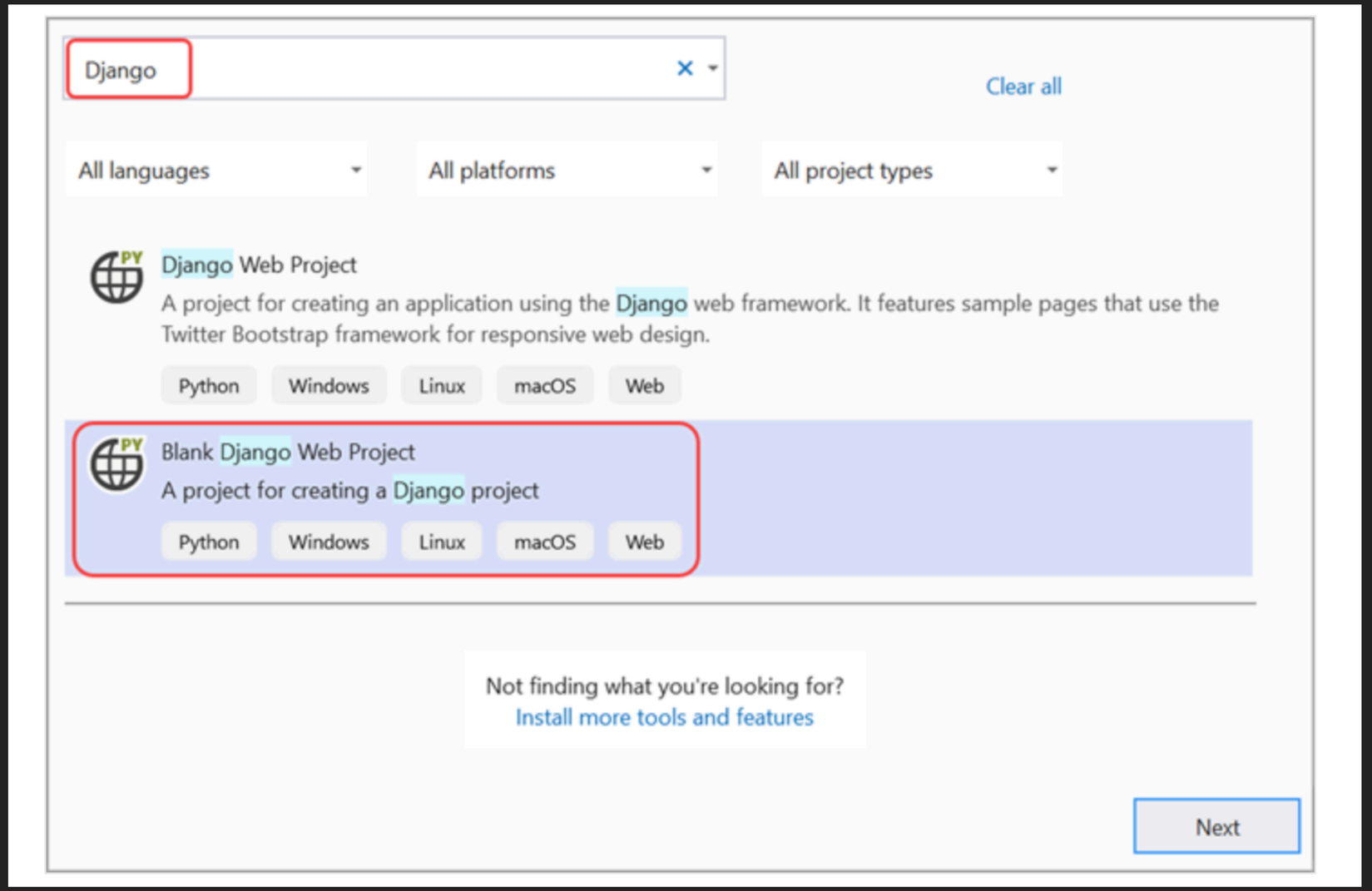
M wymusza m.in sprawdzanie wersji.

Generalnie jeżeli wszystko działa poprawnie, w terminalu powinnismy widzieć (env)$

JEŻELI Z JAKIEJŚ PRZYCZYNY MUSIELIŚMY SKORZYSTAĆ Z TERMINALA, TO TRZEBA ODPALIĆ PROJEKT

- django-admin startproject <nazwa>
- Jak udało się odpalić środowisko wirtualne w Pycharm - wówczas nie ma potrzeby.

W VISUAL STUDIO TEŻ DZIAŁA Z POZIOMU GRAFICZNEGO



W VISUAL STUDIO CODE – TEŻ LEPIEJ TERMINAL

```
# Linux
sudo apt-get install python3-venv   #jeżeli jest potrzebne
python3 -m venv .venv
source .venv/bin/activate

# macOS
python3 -m venv .venv
source .venv/bin/activate

# Windows
py -3 -m venv .venv
.venv\scripts\activate
```

W VISUAL STUDIO CODE – TEŻ LEPIEJ TERMINAL

A później:

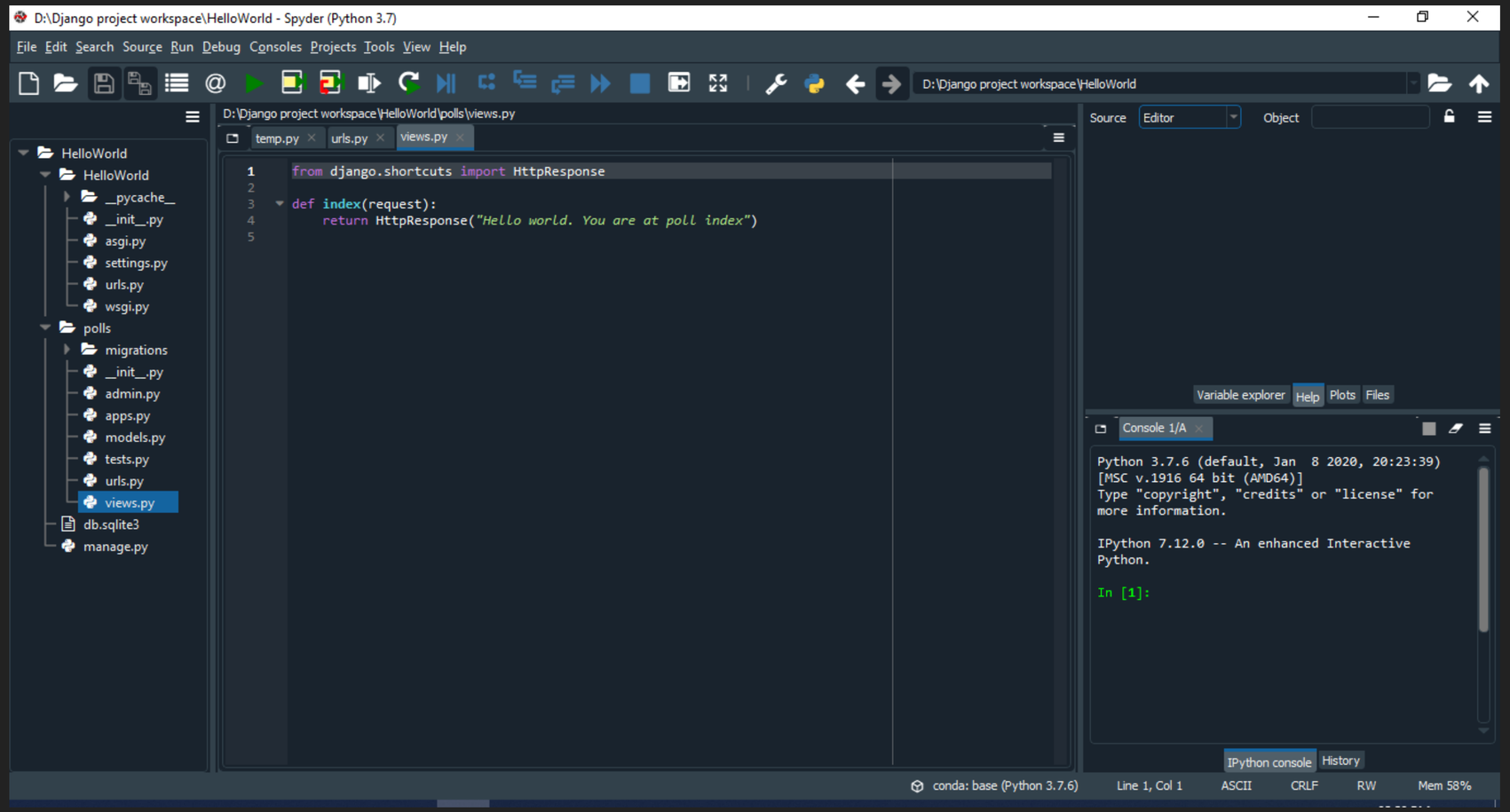
- python -m pip install django lub bez m - niby nie ma większej różnicy, ale nie zaszkodzi.

- django-admin startproject nazwa .

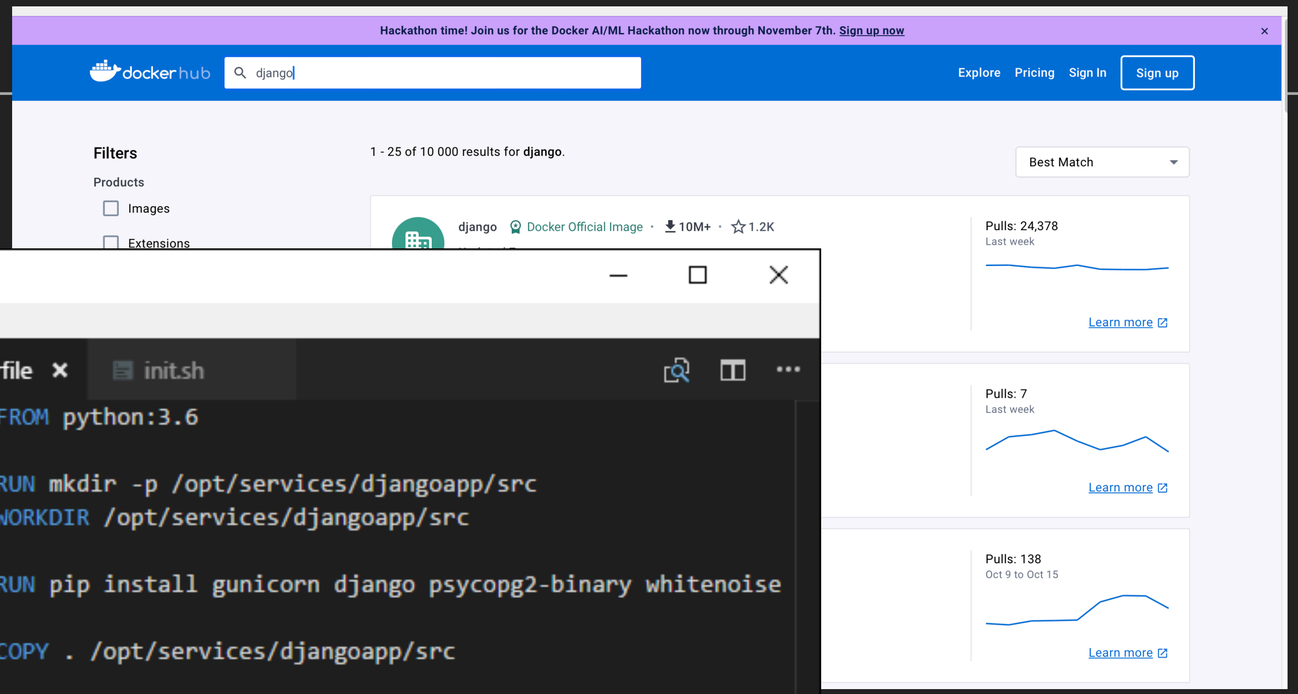
Zdaniem niektórych bardzo ważna jest ta kropka na końcu, ma sprawiać, że pracujemy w obecnym folderze. W pycharm też powinno się używać. Mi raz to działało, raz nie. Odkąd używam Pycharm komercyjnego się tym nie przejmuję.

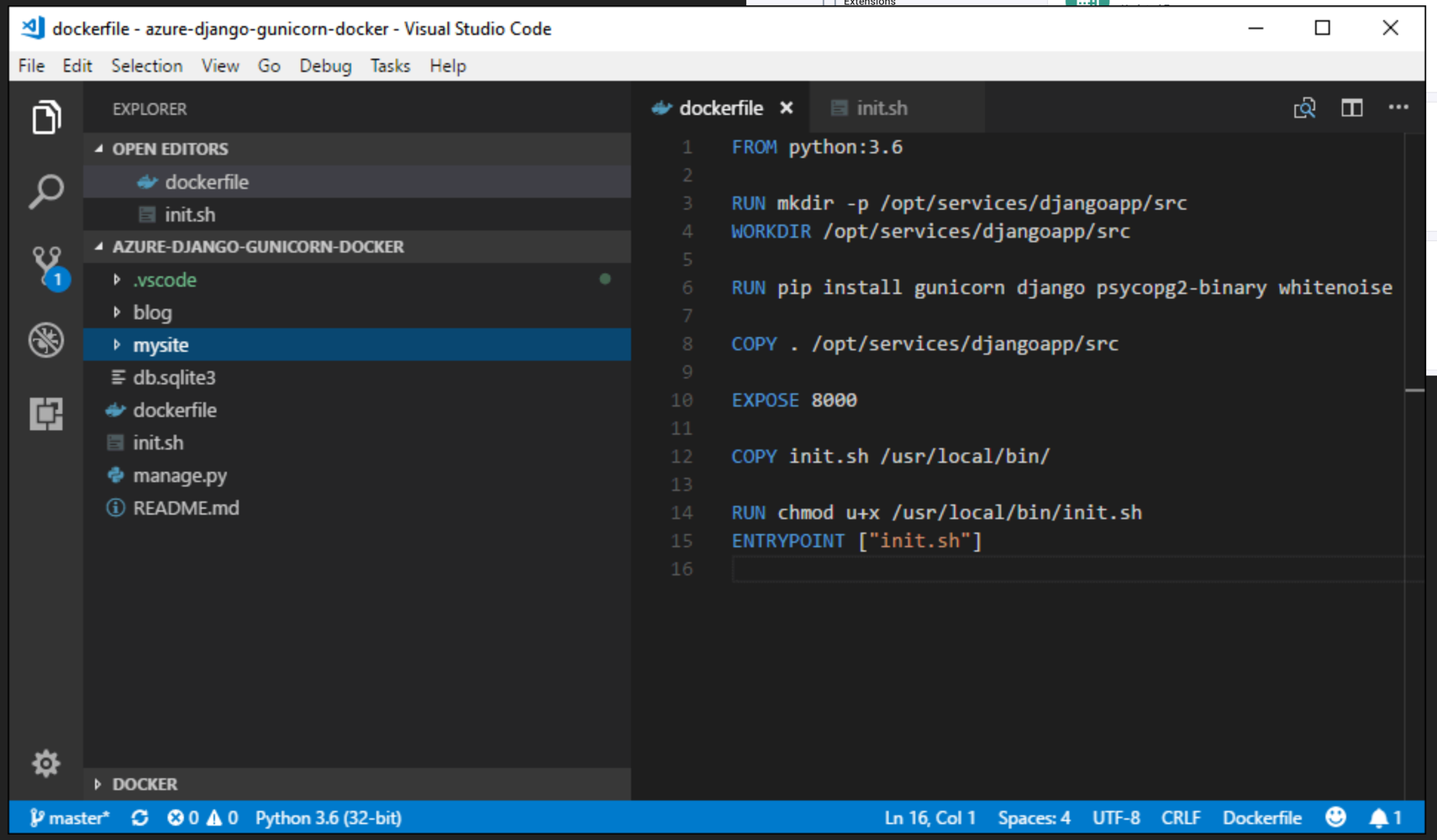
A MOŻE SPYDER

python -m pip install Django



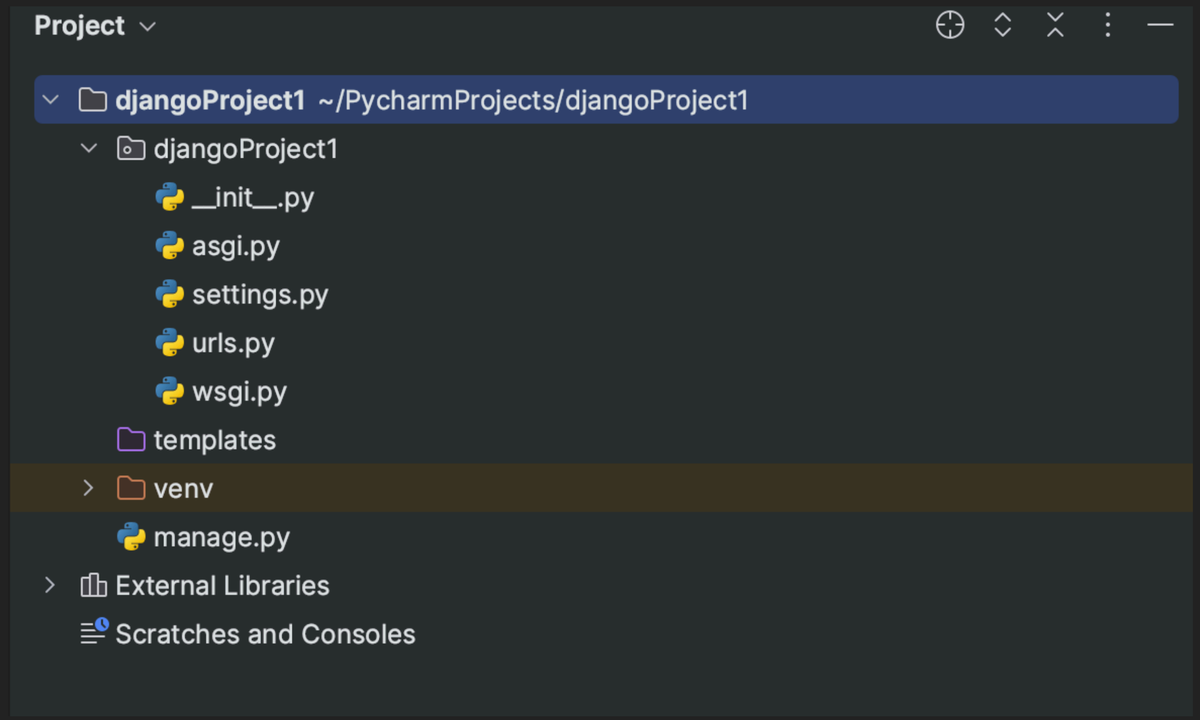
JESZCZE INNY SPOSÓB – DOCKER





STRUKTURA PLIKÓW

Dobrze by było omówić samą strukturę plików w głównym projekcie:



MANAGE.PY

Jest to plik skryptu Pythona, który służy do zarządzania projektem Django. Za jego pomocą możemy uruchamiać serwer deweloperski, tworzyć migracje bazy danych, tworzyć nowe aplikacje i wiele innych.

GENERALNIE NIE DOTYKAMY SIĘ - poza wywoływaniem komend z terminala, np.: (następny slajd)

MANAGE.PY

python manage.py runserver: Uruchamia serwer Django.

python manage.py startapp <nazwa_aplikacji>: Tworzy nową aplikację Django o podanej nazwie.

python manage.py makemigrations: Tworzy pliki migracji dla bazy danych na podstawie zmian w modelach Django.

python manage.py migrate: Wykonuje migracje bazy danych na podstawie plików migracji.

python manage.py createsuperuser: Tworzy nowego użytkownika z uprawnieniami administratora.

python manage.py collectstatic: Kopiuje wszystkie statyczne pliki z aplikacji do katalogu określonego w ustawieniach projektu.

PROJEKT/

katalog nadrzędny dla całego projektu.

Zwłaszcza właśnie w Pythonie warto przyzwyczaić się do filozofii pracy w folderach. Ale i w HTML i w wielu innych to ważne.

_INIT_PY

Ten plik jest wymagany przez Pythona, aby traktować katalog projektu jako pakiet Pythona.

SETTINGS.PY

plik zawiera ustawienia konfiguracyjne dla projektu Django, takie jak ustawienia bazy danych, ustawienia szablonów. Na początku na pewno lepiej nic nie kombinować. W zasadzie jedyną bezpieczną zmianą jest zmiana języka angielskiego na przykład na język polski.

SETTINGS.PY

DEBUG: Ustawienie to określa, czy aplikacja działa w trybie debugowania. Wartość True oznacza, że tryb debugowania jest włączony, a wartość False oznacza, że jest wyłączony. W trybie debugowania aplikacja wyświetla bardziej szczegółowe informacje o błędach i ostrzeżeniach, co ułatwia ich debugowanie. Wartość ta powinna być ustawiona na False w przypadku aplikacji produkcyjnych.

ALLOWED_HOSTS: Ustawienie to określa listę nazw hostów, które są dozwolone dla aplikacji. Domyślnie ustawione jest na [], co oznacza, że tylko lokalny adres IP jest dozwolony. W przypadku uruchamiania aplikacji na serwerze internetowym należy dodać nazwy hostów, które mają mieć dostęp do aplikacji.

DATABASES: Ustawienie to określa konfigurację bazy danych dla aplikacji. Domyślnie ustawione jest na SQLite, ale można skonfigurować aplikację do korzystania z innych baz danych, takich jak MySQL lub PostgreSQL.

STATIC_URL i STATICFILES_DIRS: Ustawienia te określają ścieżki do plików statycznych (takich jak arkusze stylów CSS i pliki JavaScript) używanych przez aplikację. Domyślnie pliki statyczne są przechowywane w katalogu static w każdej aplikacji Django.

TEMPLATES: Ustawienie to określa konfigurację silnika szablonów używanego przez aplikację. Domyślnie używany jest silnik szablonów Django.

URLS.PY

W tym miejscu zarządzamy adresami URL dla całego projektu. Można to by porównać do takiego wewnętrznego menu w HTML.

Początkowo jest tylko panel sterowania

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

ASGI.PY I WSGI.PY

Są to pliki odpowiedzialne za uruchamianie aplikacji Django na serwerze internetowym.

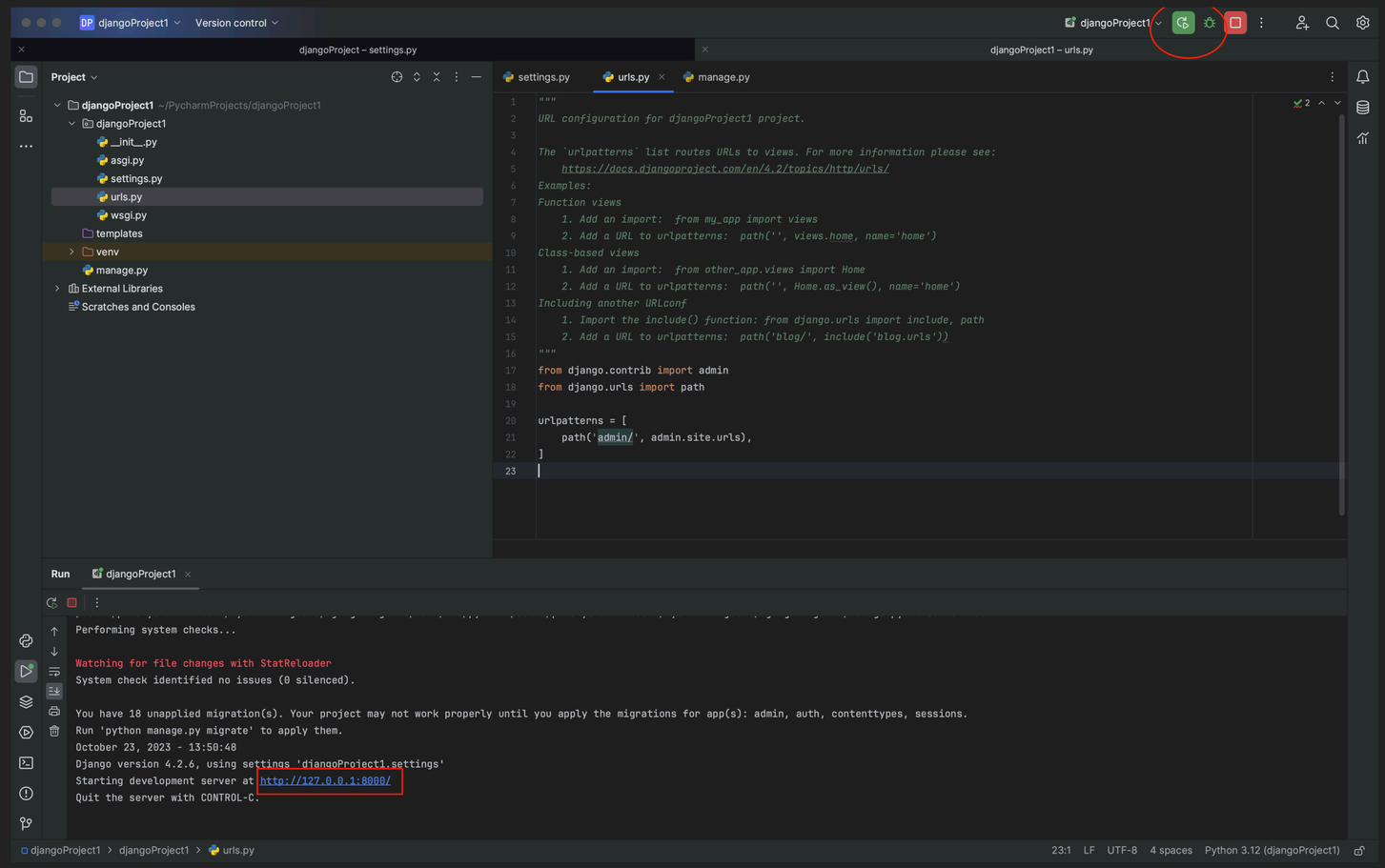
MAMY GOŁĄ INSTALACJĘ

W Pycharm, zwłaszcza komercyjnym wystarczy nacisnąć start i odpala nam się serwer z adresem IP.

W innych programach może być konieczne uruchomienie poprzez komendę:

python manage.py runserver

Aby zatrzymać - CTRL+C



Czasami po komendzie ręcznie trzeba dodać port:

python manage.py runserver 5000

APLIKACJE

W Django używamy określenia aplikacje na to wszystko, co chcemy uruchomić w projekcie. Blog, baza, komunikator - to mogą być całkiem osobne aplikacje.

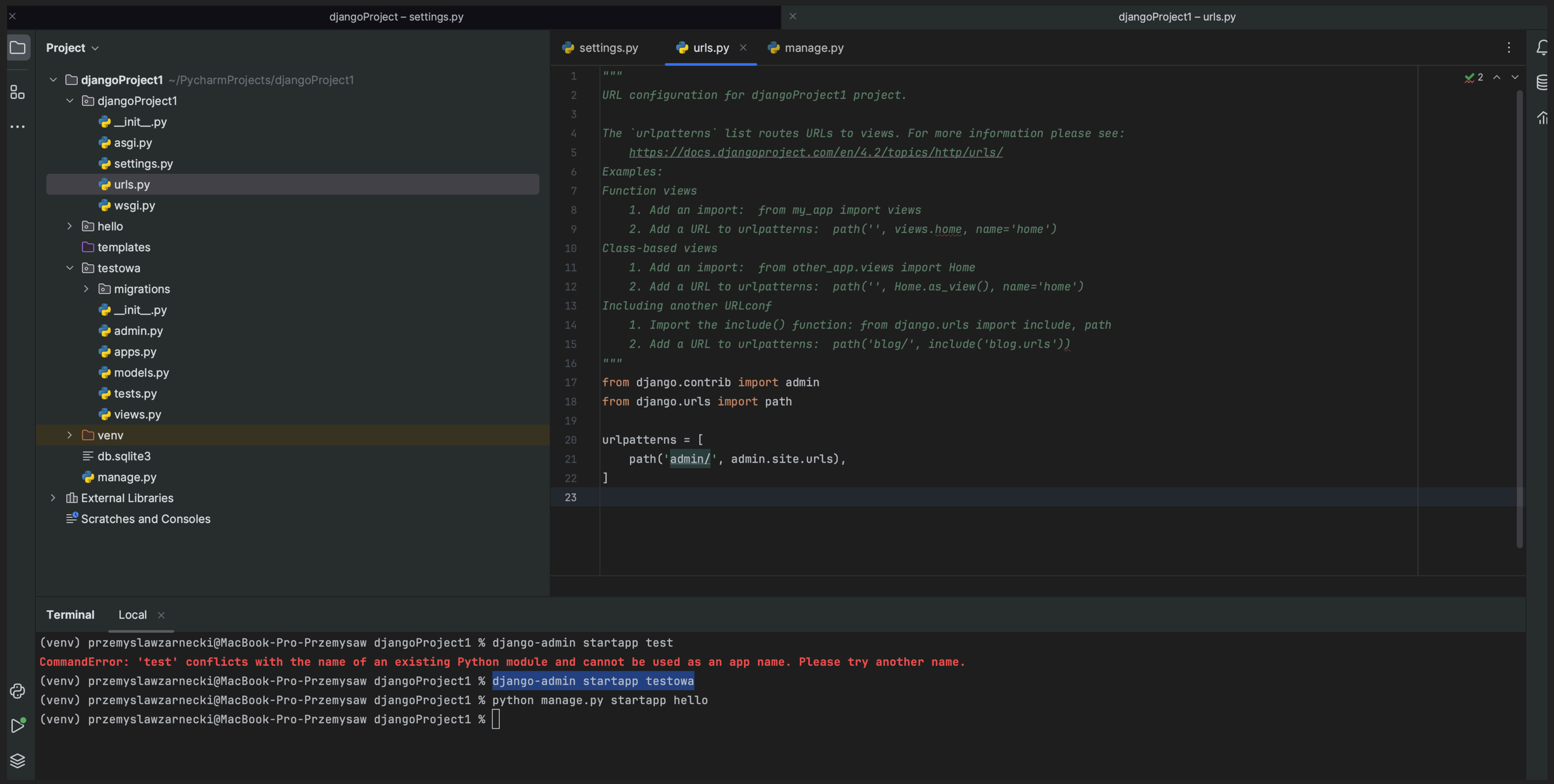
Stąd właśnie w Django ciekawie można sobie budować portfolio:

W większości poradników na start aplikacji jest komenda: python manage.py startapp NAZWA

Ja się przyzwyczaiłem do django-admin startapp NAZWA

Nie zauważyłem żadnych różnic obie działają (różnice są związane z katalogiem):

APLIKACJE


djangoProject – settings.py
djangoProject1 – urls.py
Project
djangoProject1 ~/PycharmProjects/djangoProject1
djangoProject1
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
hello
templates
testowa
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
venv
db.sqlite3
manage.py
External Libraries
Scratches and Consoles
settings.py
urls.py
manage.py
"""
URL configuration for djangoProject1 project.
The `urlpatterns` list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
urlpatterns = [
path('admin/', admin.site.urls),
]
Terminal
Local
(venv) przemyslawzarnecki@MacBook-Pro-Przemysaw djangoProject1 % django-admin startapp test
CommandError: 'test' conflicts with the name of an existing Python module and cannot be used as an app name. Please try another name.
(venv) przemyslawzarnecki@MacBook-Pro-Przemysaw djangoProject1 % django-admin startapp testowa
(venv) przemyslawzarnecki@MacBook-Pro-Przemysaw djangoProject1 % python manage.py startapp hello
(venv) przemyslawzarnecki@MacBook-Pro-Przemysaw djangoProject1 %

APLIKACJE

python manage.py migrate - tworzy plik bazy danych db.sqlite3 - do nauki ok. Dla klientów - ZEERO bezpieczeństwa

Tu może różnica pomiędzy poleceniami:

django-admin i manage.py są równoważne i wykonują to samo zadanie. Różnica między nimi polega na tym, że manage.py jest wrapperem dla django-admin, który ustawia zmienną środowiskową DJANGO_SETTINGS_MODULE i dodaje katalog projektu do sys.path. Dzięki temu możesz używać manage.py z dowolnego miejsca w projekcie, podczas gdy django-admin musi być uruchomiony z katalogu projektu.

APLIKACJE - STRUKTURA PLIKÓW

__init__.py: Ten plik jest wymagany przez Pythona, aby traktować katalog aplikacji jako pakiet Pythona.

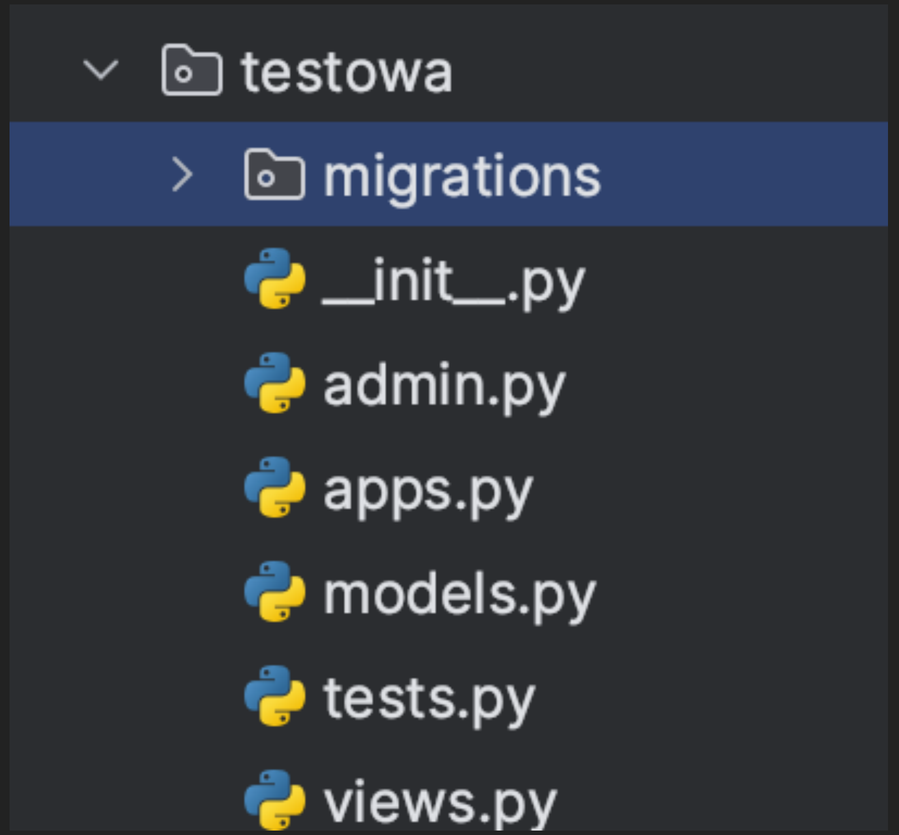
admin.py: Ten plik zawiera definicje modeli admina dla Twojej aplikacji.

apps.py: Ten plik zawiera definicję klasy AppConfig dla Twojej aplikacji.

models.py: Ten plik zawiera definicje modeli dla Twojej aplikacji.

tests.py: Ten plik zawiera testy jednostkowe dla Twojej aplikacji.

views.py: Ten plik zawiera definicje widoków dla Twojej aplikacji.

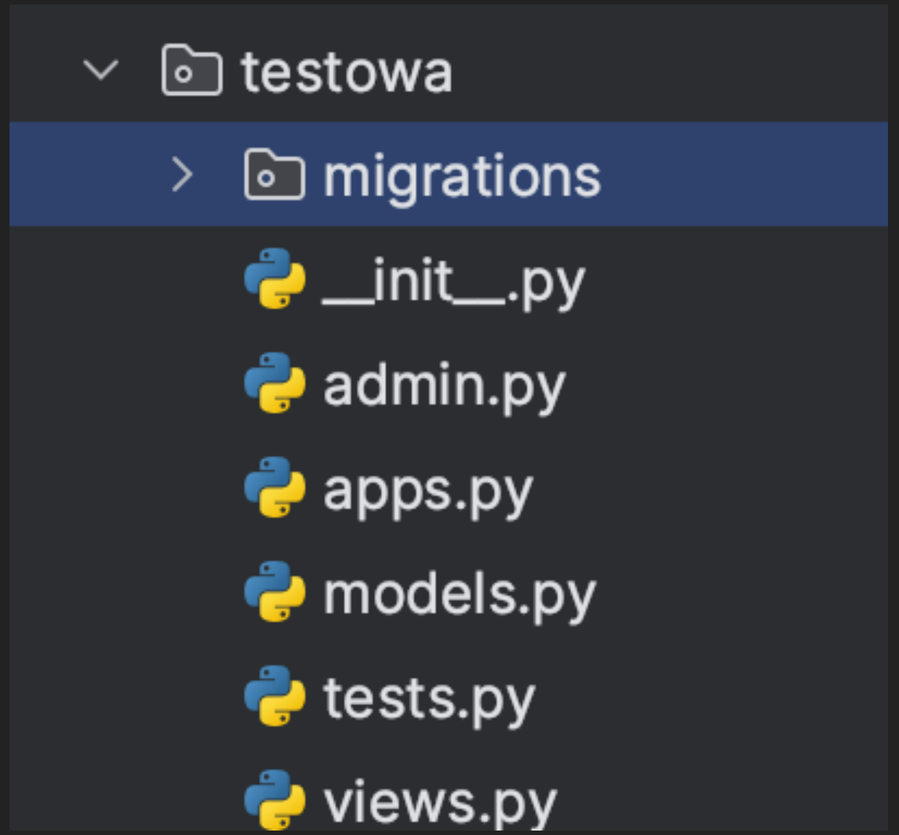


APLIKACJE - STRUKTURA PLIKÓW

Models.py zawiera definicje modeli danych dla Twojej aplikacji. Modele te reprezentują obiekty w Twojej bazie danych i definiują ich pola i zachowanie.

Z drugiej strony, admin.py zawiera definicje klas admina dla Twojej aplikacji. Klasy te pozwalają na dostęp do modeli danych z poziomu panelu administracyjnego Django. W pliku admin.py możesz zdefiniować niestandardowe widoki, formularze i filtry dla Twojego panelu administracyjnego.

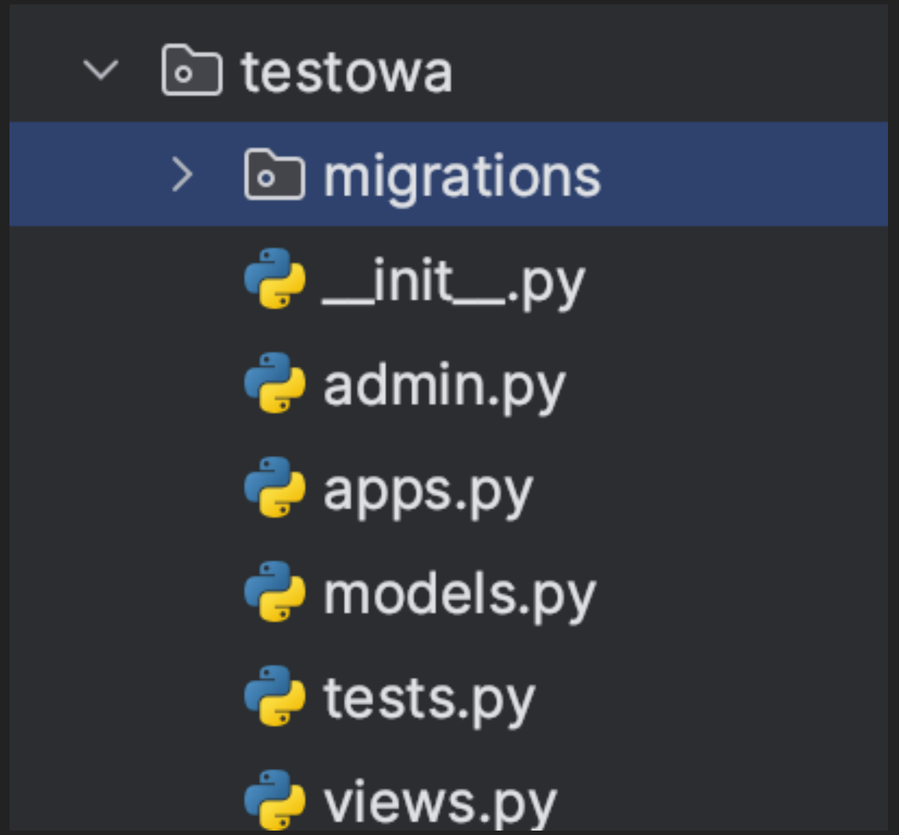
Wcześniej, w starszych wersjach Django, klasa admina była definiowana wewnątrz klasy modelu w pliku models.py. Jednakże, obecnie zaleca się umieszczanie klas admina w pliku admin.py, aby oddzielić logikę panelu administracyjnego od logiki modelu.



APLIKACJE - STRUKTURA PLIKÓW

Na początku głównie działamy w obszarze models, gdzie dodajemy elementy aplikacji. Admin, gdzie definiuje co ma się znaleźć w panelu sterowanie.

Pozostałych na początku nie ruszamy. Ale po kolei.



UTWORZYLIŚMY APLIKACJĘ I CO DALEJ

Na początku głównie działamy w obszarze models, gdzie dodajemy elementy aplikacji. Admin, gdzie definiuje co ma się znaleźć w panelu sterowanie. Oraz oczywiście widoki, czy urls.

Pozostałych na początku nie ruszamy. Ale po kolei.

UTWORZYLIŚMY APLIKACJĘ I CO DALEJ…

W **widokach** definiujemy co ma się wyświetlać - to może być zwykły html ;-)

```
from django.http import HttpResponse



def index(request):
    return HttpResponse(„Jakiś tam napis - oczywiście cudzysłów na górze„)
```

UTWORZYLIŚMY APLIKACJĘ I CO DALEJ...

W url naszej aplikacji definiujemy sobie do niej kod aby mieć do niej przekierowanie:

```
from django.urls import path


from . import views


urlpatterns = [
    path("", views.index, name="index"),
]
```

UTWORZYLIŚMY APLIKACJĘ I CO DALEJ...

Później trzeba dodać to do url projektu - aby główny projekt w ogóle to widział

```
from django.contrib import admin

from django.urls import include, path


urlpatterns = [
    path("nazwa/", include("nazwa.urls")),
    path("admin/", admin.site.urls),
]
```

UTWORZYLIŚMY APLIKACJĘ I CO DALEJ...

Dobrze by było stworzyć jakiś model - elementy aplikacji

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

UTWORZYLIŚMY APLIKACJĘ I CO DALEJ…

To wszystko trzeba wyeksportować do naszej bazy danych:

python manage.py migrate

I dalej tworzymy …